

Java Web - JSP

5: Java EE - JSP



ÍNDICE

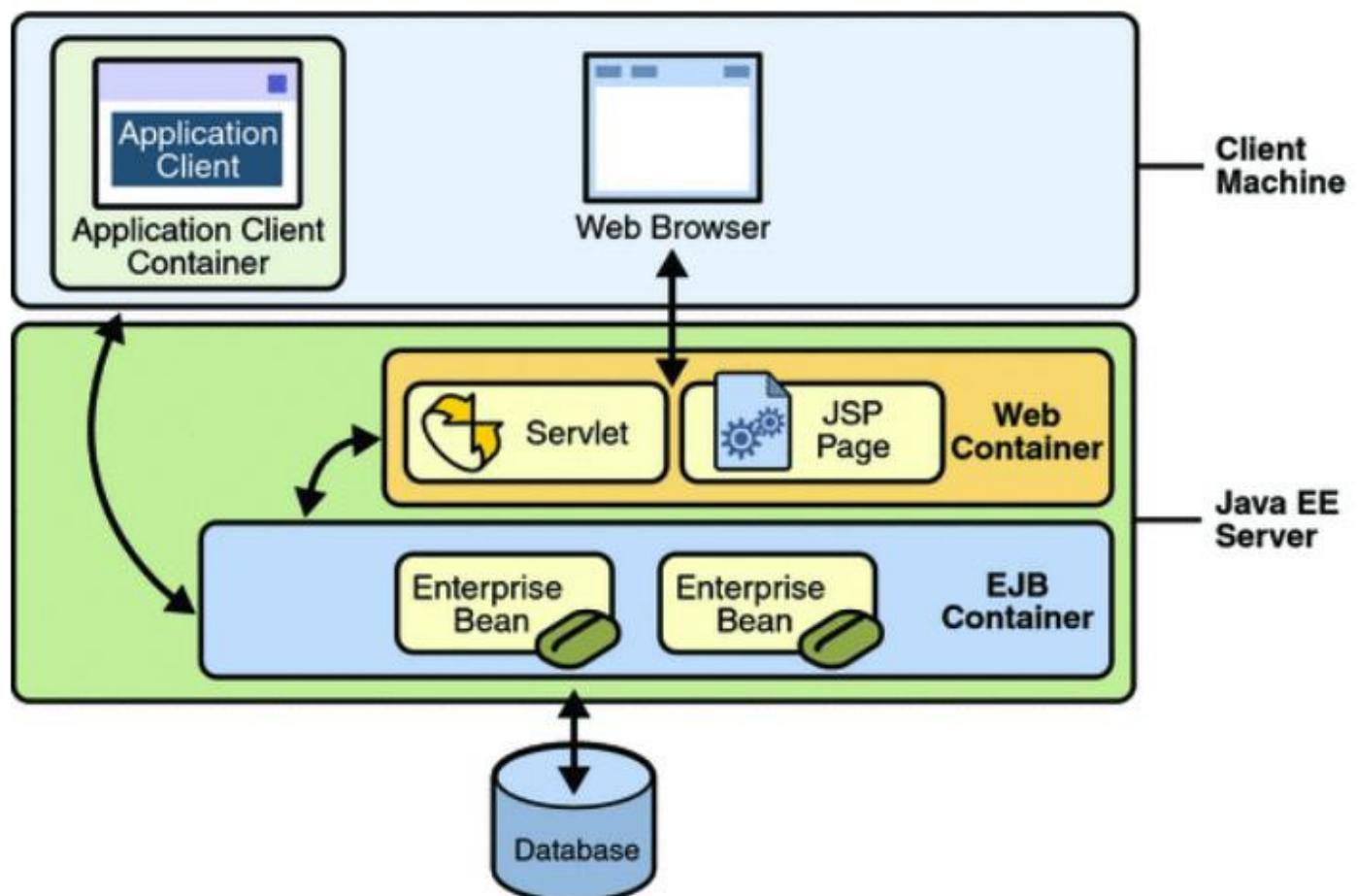
1	Java EE	2
2	Java EE - Servlets	6
3	Java EE - JavaServer Pages (JSP)	11
4	Primer Proyecto Web (PrimerProyectoWeb)	17
5	Segundo Proyecto Web (SegundoProyectoWeb)	20
6	Java EE – Arquitectura MVC	24
7	Encuesta MVC (EncuestaMVC)	27
8	JSP Standard Tag Library (JSTL)	29
9	Conexión MySQL con JSTL (PracticaJSTL)	30

1 Java EE

1.1 Java EE - Introducción

- **Java Platform, Enterprise Edition (Java EE)**
- Aplicaciones web construidas a base de **componentes**:
 - Clientes de aplicación y applets (en el cliente)
 - Java Servlet, JavaServer Faces (JSF), and JavaServer Pages (JSP) (en el servidor)
 - Enterprise JavaBeans (EJB) (o *enterprise beans*) (en el servidor)
- Los componentes se despliegan y ejecutan en **contenedores** especializados
 - Ejemplos de contenedores:
 - Contenedor de applets en un navegador Web
 - Contenedor Web Tomcat
 - Contenedor de EJBs

1.2 Java EE - Contenedores de componentes

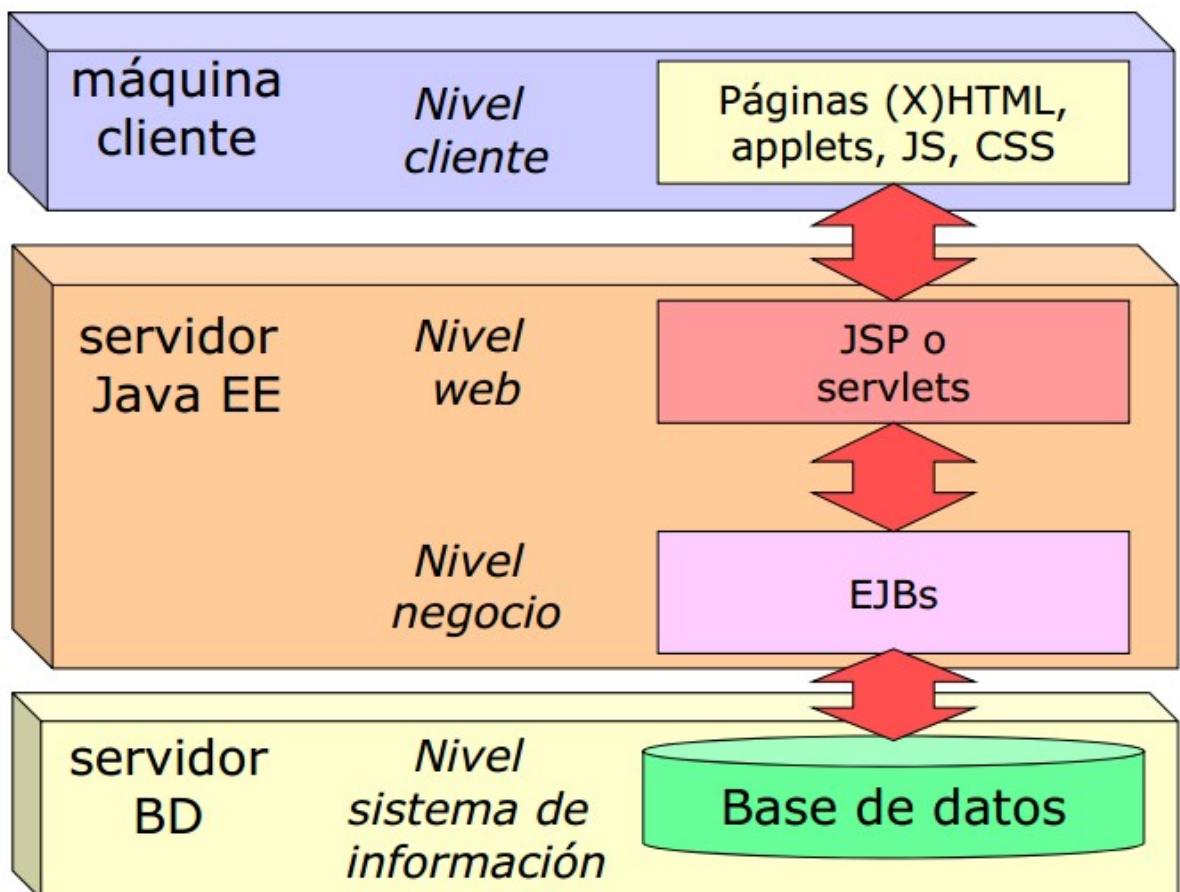


1.3 Contenedores

- Un contenedor es un proceso donde se ejecutan los componentes
 - Gestiona los componentes de la aplicación
 - **Ciclo de vida**
 - Proporciona acceso a **servicios** de la plataforma
 - Seguridad, transacciones, persistencia, conectividad, etc.
- El desarrollador tiene que especificar
 - Los componentes de la aplicación
 - Servlets
 - JSPs (JavaServer Pages)
 - JSFs (JavaServer Faces)
 - EJBs (Enterprise Java Beans)
 - Los descriptores de despliegue (*deployment*)
 - Ficheros XML que describen los componentes de aplicación

1.4 Java EE - Arquitectura multi-nivel

- Este modelo propicia aplicaciones web en 4 niveles:



1.5 Componentes web en Java EE

- **Java Servlets**
 - Clases escritas en Java que procesan peticiones HTTP y construyen respuestas
- **JavaServer Pages (JSP)**
 - Documentos basados en texto que contienen dos tipos de texto
 - una plantilla de datos estática que puede expresarse en un formato como (X)HTML o XML
 - elementos JSP que determinan cómo la página construye el contenido dinámico
- **JavaServer Faces (JSF)**
 - Componentes de interfaz de usuario para aplicaciones web
- Los **clientes** en Java EE son
 - Clientes web: navegadores web, páginas web, applets
 - Aplicaciones de cliente

1.6 Componentes de negocio en Java EE

- Lógica que resuelve las necesidades de un determinado dominio de aplicación
- **Enterprise beans (EJBs)**
 - Pueden procesar datos recibidos del lado cliente y enviarlos al nivel de sistema de información para su almacenamiento
 - Pueden recuperar datos del sistema de información, procesarlos y enviarlos al cliente
- Tipos de EJBs
 - Bean de sesión
 - Una conversación con un cliente
 - Bean dirigido por mensajes
 - Permite que un componente de negocio pueda recibir mensajes asíncronamente, normalmente con el Java Message Service (JMS)

1.7 Frameworks Java EE

- Apache Struts
- Spring
- JBoss Seam
- GWT (Google Web Toolkit)
 - Aplicaciones en Java basadas en Ajax
- Stripes, Tapestry, Wicket, Maverick, etc.

2 Java EE - Servlets

2.1 Java EE Servlets - Introducción

■ **Servlet**

- Clase Java que implementa un modelo de programación petición-respuesta
 - Puede usarse para procesar cualquier tipo de petición
 - Clase GenericServlet
 - Definido en el paquete **javax.servlet**
 - Hay subclases específicas para HTTP: paquete **javax.servlet.http**
 - Clase HttpServlet
- Tiene un ciclo de vida concreto controlado por el contenedor en el que se despliega
- Cada servlet se ejecuta como un thread independiente
- Muy útiles para leer cabeceras de mensajes, datos de formularios, gestión de sesiones, procesar información, etc.
- Pero tediosos para generar todo el código HTML
 - El mantenimiento del código HTML es difícil
- Los servlets tienen que generar todo el código HTML

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println ("<!DOCTYPE html>\n" +           "<html>\n" +
            "<head><title>Formulario de saludo</title></head>\n" +
            "<body>\n" +
            "<h1>Hola " +request.getParameter("cliente")+"</h1>\n" +
            "</body></html>");
```

2.2 Ciclo de vida de un servlet

- Lo controla el contenedor en el que se ha desplegado
- Al llegar una petición correspondiente a un servlet, el contenedor
 1. Comprueba si existe una instancia del servlet
 - Si no existe
 - Carga la clase del servlet
 - Crea una instancia del servlet
 - Inicializa la instancia del servlet llamando al método *init()*
 2. Se invoca al **método de servicio**, pasándole objetos de tipo *request* y *response*
 - El servlet usa estos objetos para inspeccionar la petición y generar la respuesta
 3. Si hay que eliminar el servlet, el contenedor llama al método *destroy()* del servlet

2.3 Servlet

- Cualquier clase que implemente la interfaz **javax.servlet.Servlet**
 - Métodos para gestionar el ciclo de vida del servlet
- También suele ser necesaria la interfaz **javax.servlet.ServletConfig**
 - Tiene los parámetros de arranque e inicialización para el servlet que le pasa el contenedor
- Normalmente los servlets se implementan a partir de una de las dos subclases siguientes que implementan ambas interfaces:
 - **javax.servlet.GenericServlet**
 - Clase que define un servlet independiente del protocolo
 - **javax.servlet.http.HttpServlet**
 - Para servlets en aplicaciones web, que procesan las peticiones con el protocolo HTTP

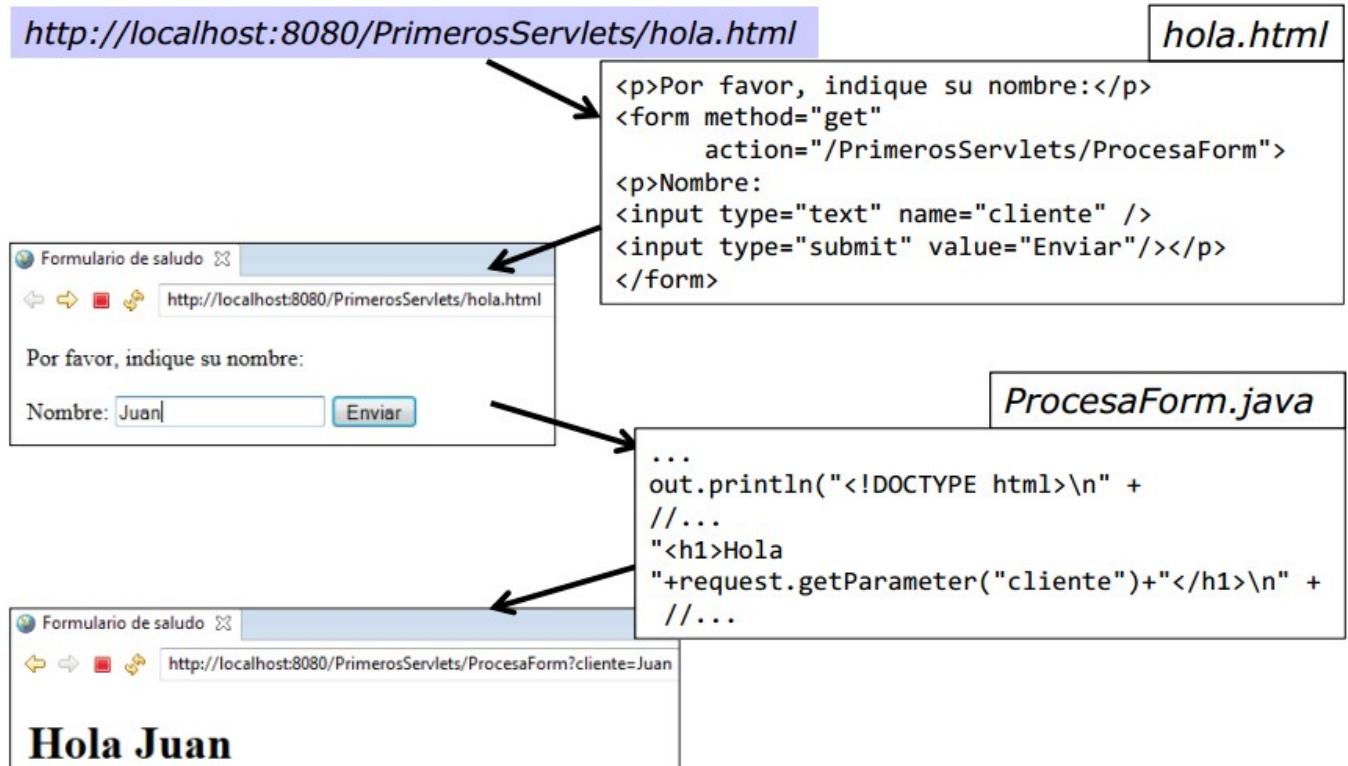
2.4 Métodos de la interfaz javax.servlet.Servlet

- **init(ServletConfig config)**
 - Al arrancar un servlet, solo se ejecuta una vez
 - Inicializa atributos y recursos del servlet
- **getServletConfig()**
 - Devuelve el objeto *ServletConfig* que contiene parámetros de arranque e inicialización para el servlet que le pasa el contenedor
- **service(ServletRequest request, ServletResponse response)**
 - Cada vez que se invoca un servicio al servlet
 - Dependiendo del tipo de servicio, invoca el método correspondiente al servicio solicitado
 - Normalmente no se reescribe este método, solo los correspondientes a los servicios específicos
 - Pueden invocarse concurrentemente los métodos de servicio, por ello deben ser thread safe
- **destroy()**
 - Al eliminar un servlet, solo se ejecuta una vez
 - Tiene que ser thread-safe porque puede haber otros threads en ejecución

2.5 Métodos de la clase javax.servlet.HttpServlet

- El método **service** trata las siguientes peticiones:
 - DELETE, GET, HEAD, OPTIONS, POST, PUT, y TRACE
 - Invoca el método correspondiente **doXXX**
- Los más usados:
 - **doGet**(HttpServletRequest request, HttpServletResponse response)
 - **doPost**(HttpServletRequest request, HttpServletResponse response)
- Estos dos métodos son los que normalmente se tienen que reescribir
 - El objeto request tiene información enviada en la petición
 - El objeto response permite crear la respuesta
 - `setContentType()` – permite indicar el tipo MIME de lo que se va a generar (por ejemplo, "text/html")
 - `getWriter()` – devuelve el PrintWriter donde escribir lo que se genere
 - Pueden producir dos excepciones:
 - IOException, porque usan operaciones de E/S
 - ServletException, cualquier otra excepción que se quiera

2.6 Interacción con formularios



2.7 Ejemplo de proceso de formulario

```
package es.ucm.prueba;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/ProcesaForm")
public class ProcesaForm extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println ("<!DOCTYPE html>\n" + "<html>\n" +
                    "<head><title>Formulario de saludo</title></head>\n" +
                    "<body>\n" +
                    "<h1>Hola " +request.getParameter("cliente")+"</h1>\n" +
                    "</body></html>");
    }
}
```

2.8 Parámetros de la petición

- Los parámetros que llegan con GET y POST los tiene el objeto `request` y se pueden acceder con varios métodos:
 - `public Enumeration getParameterNames()`
 - Devuelve un `Enumeration` con los nombres de los parámetros
 - Si no los hubiera, el `Enumeration` estará vacío
 - `public String getParameter(String name)`
 - Valor del parámetro como un `String`
 - `null` si no existiera ese parámetro
 - `public String[] getParameterValues(name)`
 - Si un parámetro puede tener varios valores mejor usar este método

- Un objeto `Enumeration` se usa como un Iterador:

```
Enumeration e = request.getParameterNames();
while (e.hasMoreElements()) {
    out.println(e.nextElement());
}
```

- Para pasar los valores de los parámetros de String al valor del tipo correspondiente
 - int n = new Integer(parametro).intValue();
 - double d = new Double(parametro).doubleValue();
 - byte b = new Byte(parametro).byteValue();
 - Estas operaciones se tienen que hacer en un try para poder capturar la excepción NumberFormatException
 - boolean p = new Boolean(param).booleanValue();
- Hay que comprobar siempre que los parámetros recibidos son correctos
 - Si falta un campo en el formulario, getParameter() devuelve null
 - Hay que comprobar que el string tiene el formato correcto
 - Por ejemplo, que al convertir a un número no se genera la excepción *NumberFormatException*
 - Cuando no se ha llenado el campo correspondiente se recibe el string vacío
 - Conviene eliminar los espacios en blanco con el método **trim()**

```

String parametro = request.getParameter("nombre");
if ((parametro == null) || (parametro.trim().equals("")))) {
    // Tratamiento de parámetro erróneo
}
else {
    // Tratamiento normal con el parámetro
}

```

- Lo habitual cuando hay errores en los formularios es volver a mostrarlos de nuevo
 - Mostrando los valores correctos en sus campos (el usuario no tiene que volver a introducirlos)
 - Marcar los campos que no se han llenado
 - Si se han llenado mal, añadir un comentario para indicar al usuario qué tiene que hacer
- Con JSF, o frameworks como Struts, es más fácil la gestión de formularios
- También es conveniente usar JavaScript para hacer una primera comprobación de los campos del formulario

3 Java EE - JavaServer Pages (JSP)

3.1 Java EE JavaServer Pages (JSP) - Introducción

■ JSP (JavaServer Pages)

- Fichero con código (X)HTML que incluye scripts codificados en Java
 - Permite usar (X)HTML para definir gran parte de la página
 - E introducir código Java en las partes dinámicas de la página
 - Mediante etiquetas especializadas (*Custom Tags*) que amplían la sintaxis de HTML
- Se compila y se convierte en un servlet (solo la primera vez que se invoca)
 - Se ejecuta como un servlet
- Con JSP es más fácil que se distribuya la tarea de diseño de la página web y la programación de la aplicación web
- Las JavaServer Pages (JSP) permiten escribir código HTML e insertar código Java para las partes dinámicas

3.2 JSP

- El texto HTML se denomina plantilla
- Los ficheros JSP deben tener la extensión .jsp
 - Se traducen en un servlet, que será compilado automáticamente
 - Se ejecutará el servlet generado
 - El cliente no
- En eclipse se crean dentro de WebContent
 - Igual que los ficheros .html
- El código Java se enmarca de varias maneras:
 - **<%= expresión %>**
 - El resultado de evaluar la expresión se inserta en la página HTML
 - **<% código %>**
 - Un **scriptlet**
 - El código se insertará en el método de servicio del servlet
 - **<%! declaraciones %>**
 - Las declaraciones se insertan en la clase, no en un método
 - **<%-- Comentario --%>**
 - Comentario JSP

3.3 Objetos predefinidos

- **request**
 - Objeto HttpServletRequest que permite acceder a la información de la solicitud
- **response**
 - Objeto HttpServletResponse para generar la respuesta
- **session**
 - Objeto HttpSession asociado a la petición
 - Si no hubiera sesión será null
- **out**
 - Objeto JspWriter (similar a un PrintWriter) para generar la salida para el cliente
- **application**
 - El objeto ServletContext del contenedor web

3.4 Expresiones

- **<%= expresión Java %>**
 - El resultado de evaluar la expresión será un String que pasará a formar parte de la página HTML
 - Se genera un servlet donde el resultado de la expresión se pone como `out.println(expresión)` dentro del método `_jspService()`

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
        <title>Hora</title>
    </head>
    <body>
        <p>Hora actual: <%= new java.util.Date() %></p>
        <p>Tu IP: <%= request.getRemoteHost() %></p>
    </body>
</html>
```

3.5 Scriptles

- Es la forma de insertar código Java en JSP
 - Entre los símbolos <% y %>
 - Este código se insertará tal cual en el método `_jspService()` del servlet generado
 - En el scriptlet, el texto a generar de la página HTML tiene que ponerse con `out.print()`
- Normalmente es más práctico usar scriptlets que expresiones
 - Muchas veces hay que comprobar valores, realizar acciones más complejas, etc.
 - Por ejemplo, en vez de la expresión siguiente

```
<p>Autor = <%= application.getInitParameter("Autor") %></p>
```
 - Mejor el scriptlet:

```
<%  
String autor = application.getInitParameter("Autor");  
if ((autor == null)|| (autor.trim().equals("")))  
    autor = "Anónimo";  
out.println("Autor = "+autor);  
%>
```

3.6 Partes condicionales

- Con scriptles se puede condicionar la ejecución de partes del fichero JSP
 - No obstante este mecanismo puede dar lugar a código poco claro

```
<p>  
<% String parametro = request.getParameter("nombre");  
if ((parametro == null) || (parametro.trim().equals(""))) { %>  
No nos has dado tu nombre.  
<% } else { %>  
Bienvenido,  
<% out.println(parametro); } %>  
</p>
```

3.7 Declaraciones

- Se pueden incluir declaraciones en la clase del servlet generado con <%! declaración %>
 - Este código se inserta fuera de los métodos de la clase, como nuevas declaraciones en la clase
 - Variables del servlet

```
<%! private int edad; %>
```

 - Si se declaran variables con <% ... %> serán locales al scriptlet
 - Métodos
 - Es mejor declararlo en una clase Java aparte

```
<%! private int contador = 0; %>
<p>Número de veces que se ha visitado esta página desde que se
arrancó el servidor:
<%= ++contador %>
</p>
```

3.8 Directivas

- Se aplican a la clase servlet generada
- Sintaxis:

```
<%@ directiva atributo="valor" %>
```

o bien:

```
<%@ directiva atributo1="valor1"
                     atributo2="valor2"
                     ...
                     atributoN="valorN" %>
```
- Directivas comunes
 - **include** – permite incluir otro fichero que se tratará como JSP
 - Puede tratarse de un fichero JSP, HTML, JavaScript, etc.
 - El fichero se referencia con una URL relativa a la página JSP o al servidor si va precedido de /
 - **page** – permite importar un paquete

```
<%@ page import="java.util.*" %>
```

3.9 Java Beans

- Los Java Beans son componentes Java que se usan habitualmente en aplicaciones Web para gestionar la lógica de negocio
- Se pueden utilizar en JSP para obtener información a visualizar
 - Más adelante se verá la arquitectura MVC donde JSP implementa la vista
- Clases Java que cumplen varias convenciones
 - **Declarados dentro de un paquete**
 - **Constructor sin argumentos**
 - O que no se defina ningún constructor
 - Todos los atributos son **private**
 - Estos atributos se denominan **propiedades**
 - Métodos de acceso a las propiedades
 - **getPropiedad()** para lectura
 - Para los booleanos **isPropiedad()**
 - **setPropiedad(valor)** para escritura
 - Métodos para realizar funciones más complejas
- Ejemplo de un Bean cliente

```
public class Cliente {  
    private String nombre;  
    private String nif;  
    private String email;  
    private String direccion;  
    private String telefono;  
  
    public String getTelefono() {  
        return telefono;  
    }  
    public void setTelefono(String telefono) {  
        this.telefono = telefono;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public String getDireccion() {  
        return direccion;  
    }  
    public void setDireccion(String direccion) {  
        this.direccion = direccion;  
    }  
    public String getNif() {  
        return nif;  
    }  
    public String getEmail() {  
        return email;  
    }  
}
```

3.10 JSP y Java Beans

- JSP proporciona varias etiquetas para usar Java Beans

- **jsp:useBean**

- Crea un Java Bean

```
<jsp:useBean id="nombreBean" class="paquete.Clase" />
```

- Equivale a

```
<% paquete.Clase nombreBean = new paquete.Clase(); %>
```

- **jsp:setProperty**

- Modifica una propiedad llamando al método setPropiedad()

```
<jsp:setProperty name="nombreBean"
```

```
    property="propiedad" value="valor" />
```

- Equivale a

```
<% nombreBean.setPropiedad("valor"); %>
```

- **jsp:getProperty**

- Obtiene el valor de una propiedad llamando a getPropiedad()

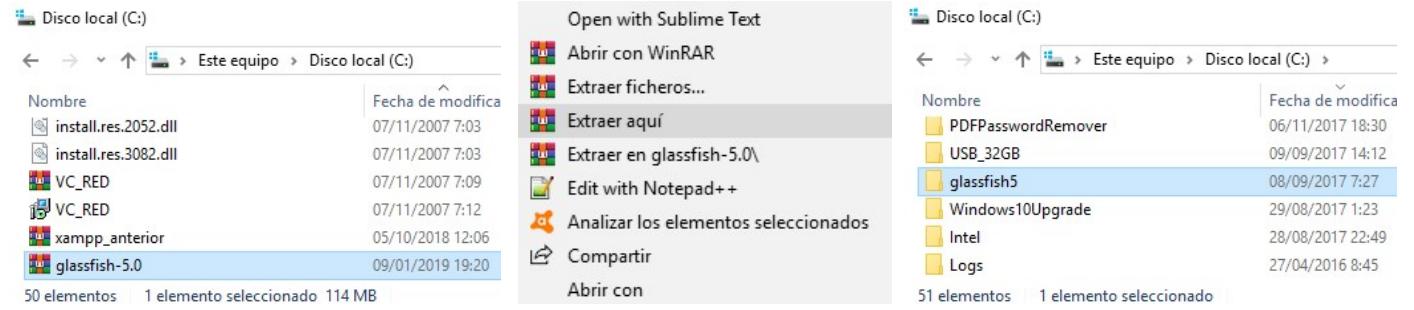
```
<jsp:getProperty name="nombreBean" property="propiedad" />
```

- Equivale a

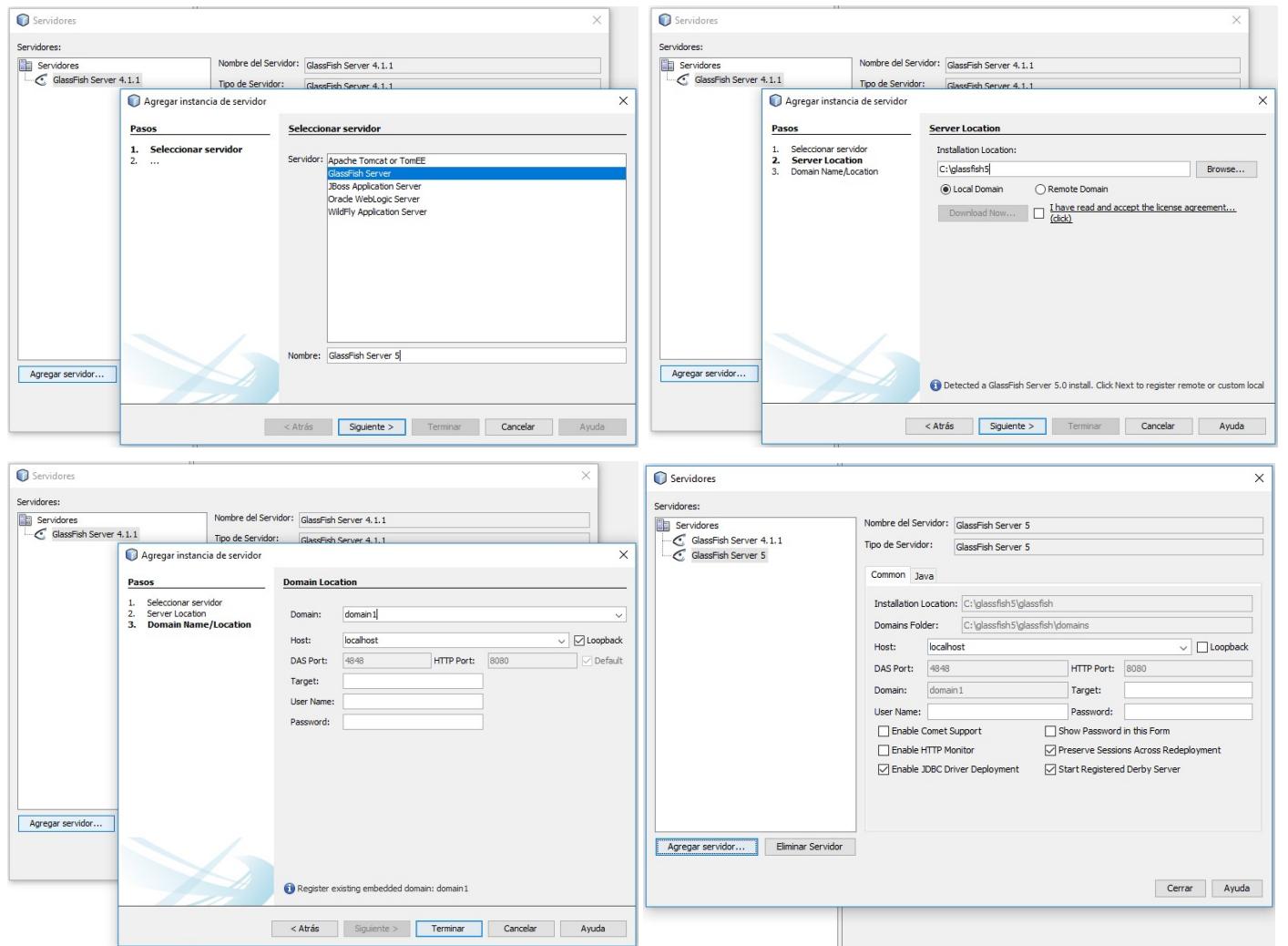
```
<%= nombreBean.getPropiedad() %>
```

4 Primer Proyecto Web (PrimerProyectoWeb)

Lo primero que debemos hacer, es instalar una nueva versión del servidor **GlassFish**. Descargamos el zip de **glassfish-5.0** en el directorio raíz del ordenador y lo descomprimimos ahí.



En NetBeans, desde el menú “**Herramientas**”, elegimos “**Servidores**”, y le damos a “**Agregar servidor...**”. Le añadimos al nombre un “**5**”, para destacar la nueva versión, y mantenemos las opciones por defecto.



Para evitar conflictos con otras herramientas utilizadas en clase, deberemos cambiar el puerto, y como no se permite realizar el cambio desde aquí, lo haremos manualmente modificando un fichero directamente.

Editamos el fichero “**C:\glassfish5\glassfish\domains\domain1\config\domain.xml**”.

Buscaremos “**8080**”, una vez encontrado, lo cambiamos por “**9999**” y guardamos los cambios.

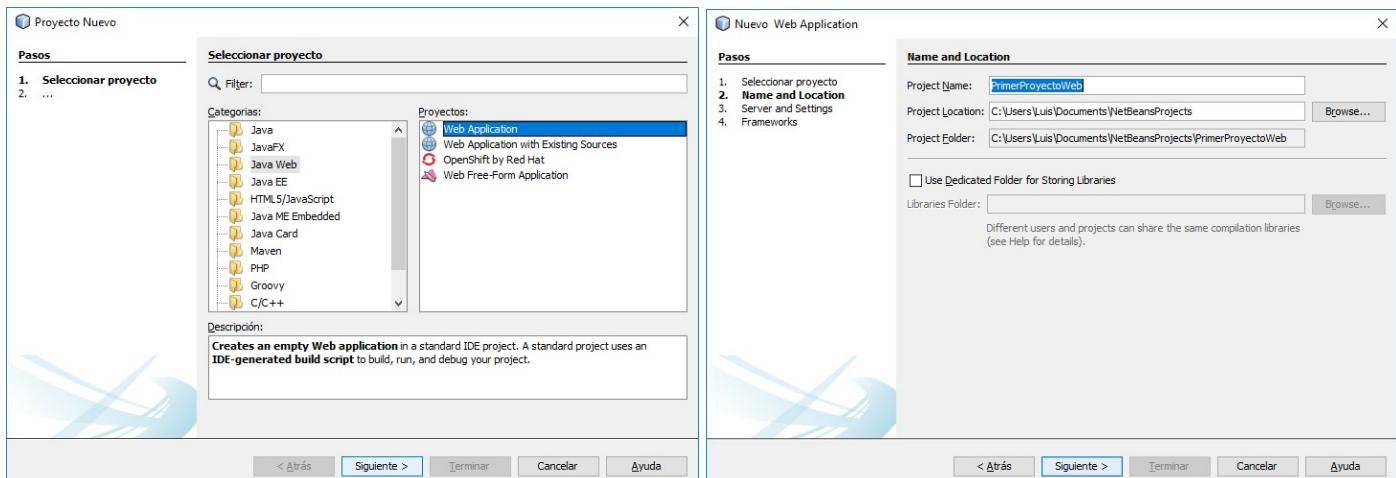
```

C:\glassfish5\glassfish\domains\domain1\config\domain.xml - Notepad++
Archivo Editar Buscar Vista Codificación Lenguaje Configuración Herramientas Macro Ejecutar Plugins Ventana ?
Buscar Reemplazar Buscar en archivos Marcar
Buscar: 8080 Siguiente >
Contar
Buscar en todos los archivos abiertos
Buscar solo en el archivo actual
Cerrar
Transparencia
Al perder el foco
Siempre
Find: Found the 1st occurrence from the top. The end of the document has been reached.
</protocols>
<network-listeners>
    <network-listener protocol="http-listener-1" port="8080" name="http-listener-1">

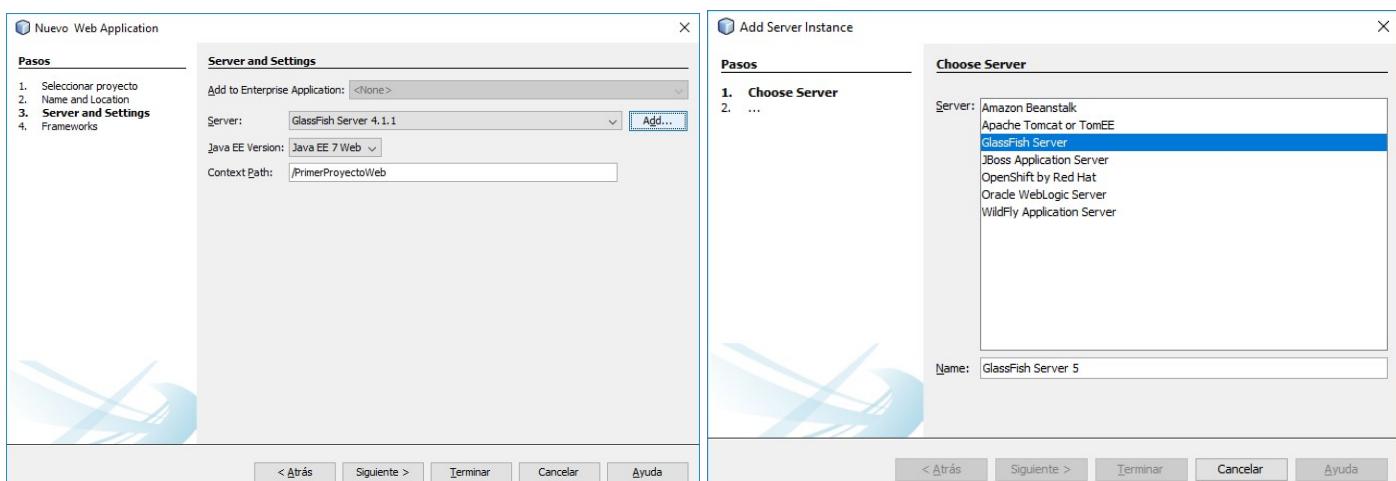
```

Cuando elegimos proyecto nuevo seleccionamos la categoría “Java Web”, y en el cuadro de al lado, dejamos la primera opción, “Web Application”. Después pulsamos en “Siguiente”.

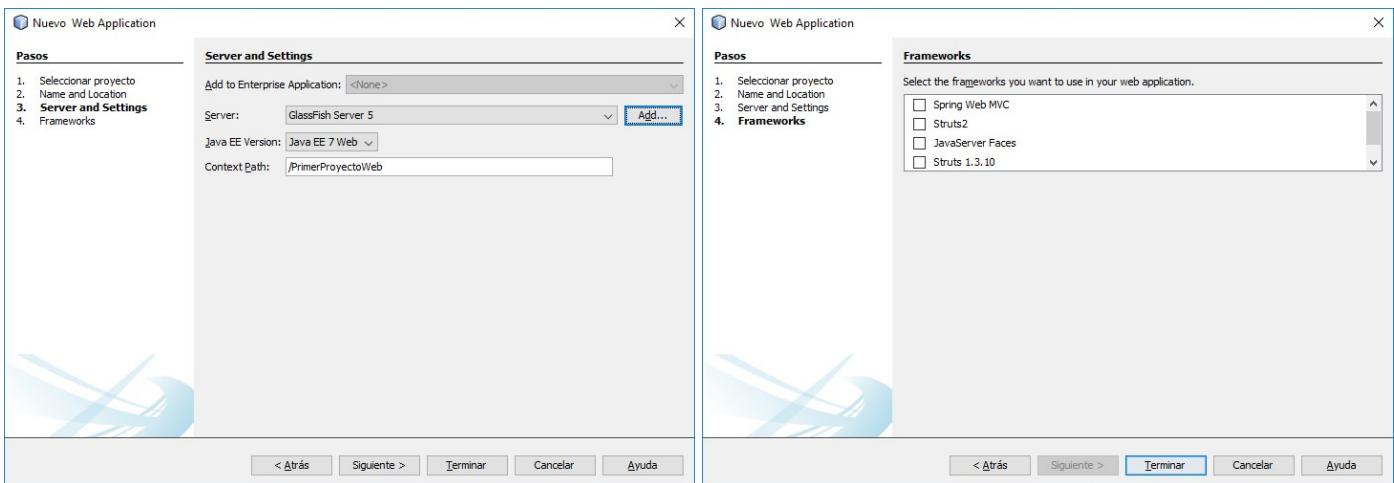
En la siguiente pantalla, ponemos el nombre del proyecto, en nuestro caso, “PrimerProyectoWeb”, y dejamos las demás opciones por defecto. Cómo antes, pulsamos en “Siguiente”.



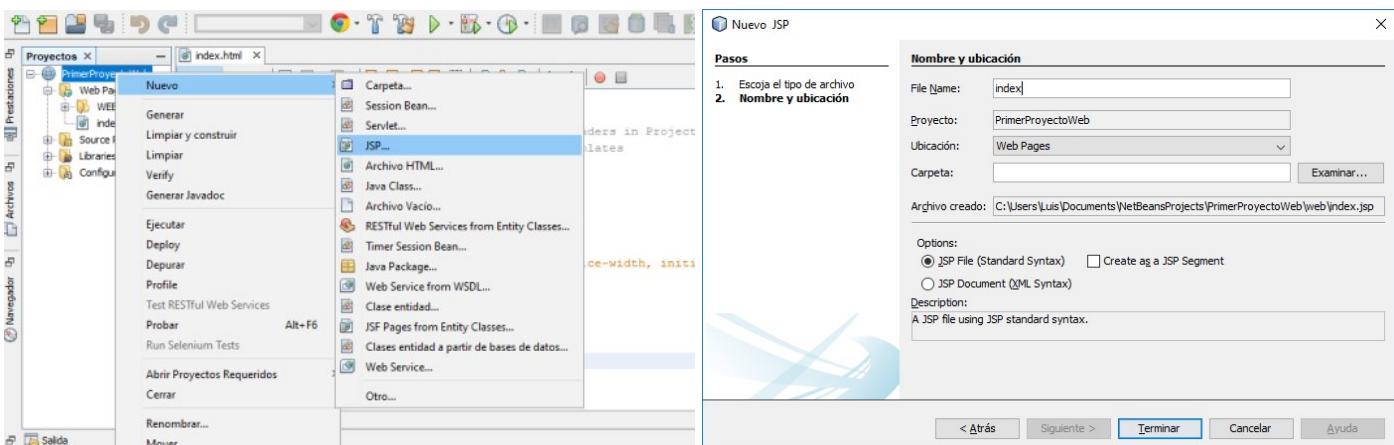
En la pantalla de servidor web si aparece por defecto “GlassFish Server 4.1.1”, lo cambiamos a “GlassFish Server 5”, o lo añadimos con el botón “Add”. Elegimos “GlassFish Server”, le añadimos “5” al nombre, y “Siguiente”.



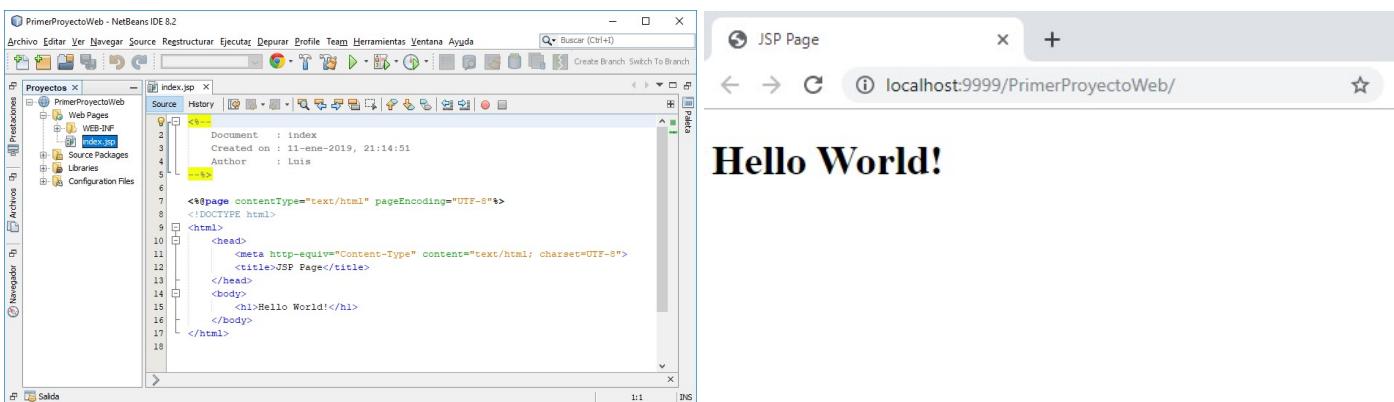
Con el servidor completado, pulsamos en “Siguiente”, y como no seleccionamos ningún Frameworks, a “Terminar”.



Con el proyecto creado, lo seleccionamos, ponemos el Chrome de navegador y creamos un fichero nuevo eligiendo desde el menú contextual “Nuevo”, “JSP”, de nombre “index” y pulsamos “Terminar”.

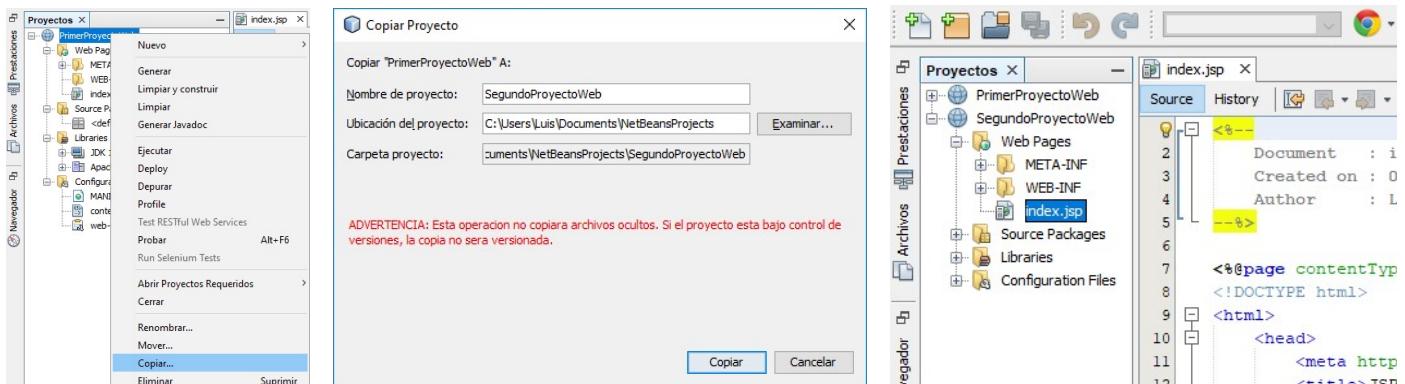


Como no nos hace falta, borramos el “index.html” y ejecutamos el proyecto.

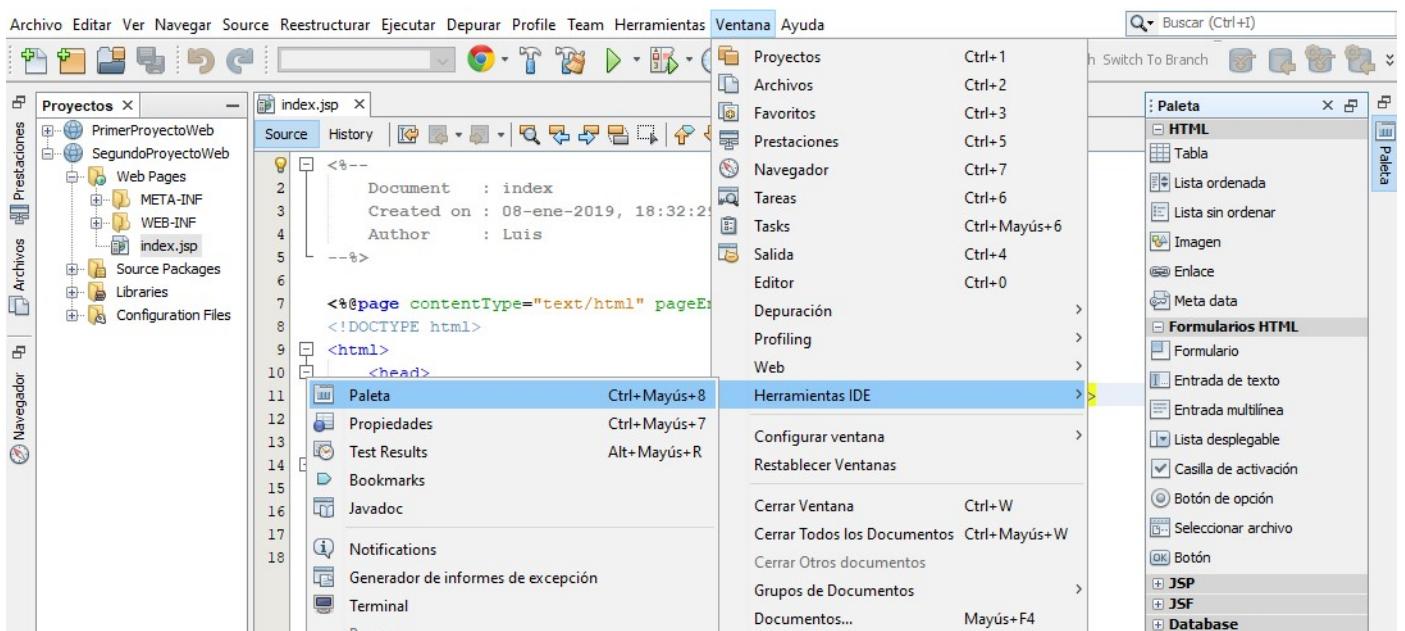


5 Segundo Proyecto Web (SegundoProyectoWeb)

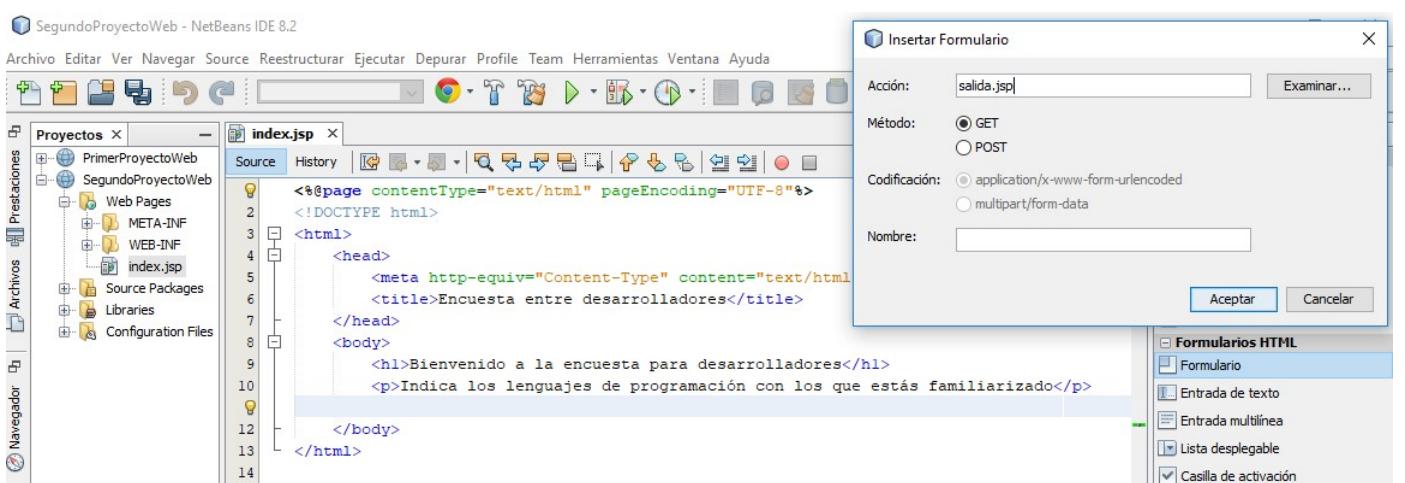
Continuamos desde el proyecto anterior seleccionando el nombre, y desde el menú contextual, elegimos “Copiar”. Cambiamos el nombre y ponemos “SegundoProyectoWeb” y pulsamos el botón “Copiar”. Ahora podemos cerrar el fichero “index.jsp” del primer proyecto, y comprimir su rama de contenidos. Seleccionamos el nombre del segundo proyecto que acabamos de copiar, volvemos a poner el Chrome de navegador y abrimos el fichero “index.jsp”.



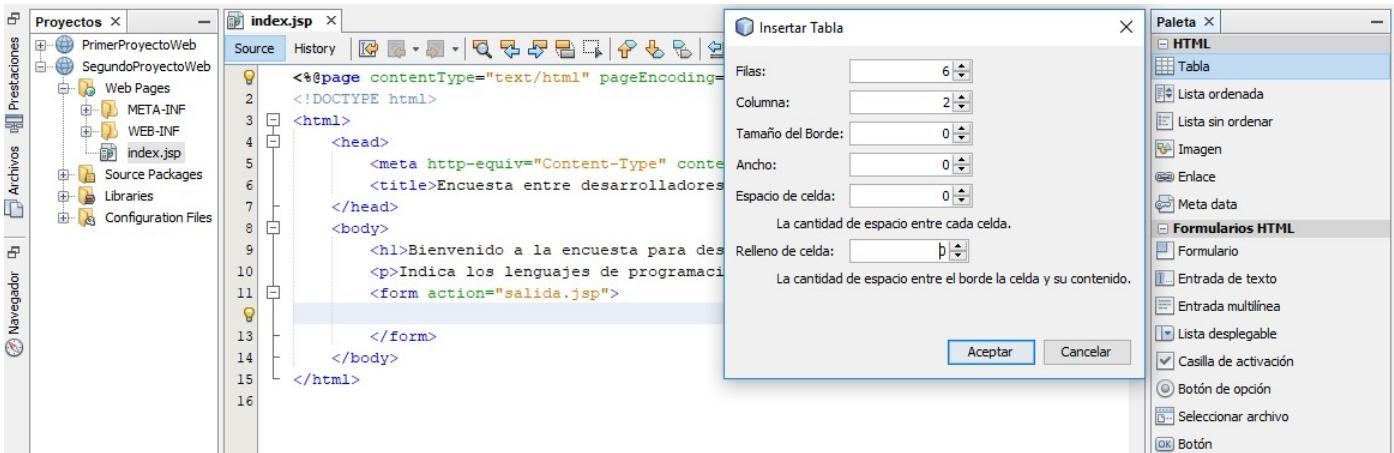
Antes de modificar el fichero index.jsp, vamos a mostrar una paleta de elementos que podemos arrastrar para utilizarlos fácilmente con asistentes, desde el menú “Ventana”, “Herramientas IDE” y “Paleta”.



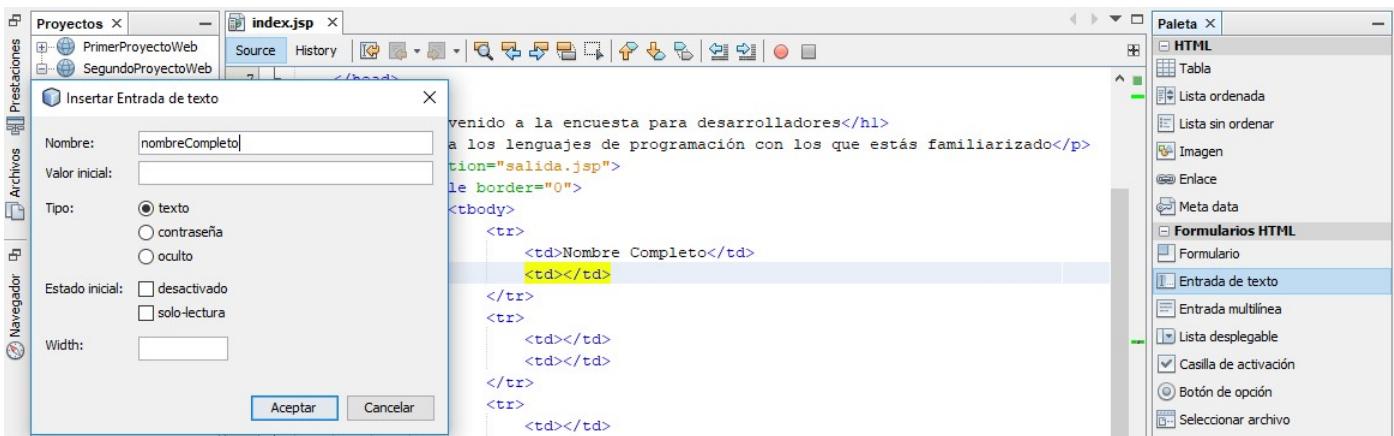
Modificamos “index.jsp”, borramos los comentarios, modificamos “<title>” y “<h1>”, añadimos “<p>” y un “Formulario” arrastrando el componente de la paleta, ponemos el nombre de “salida.jsp” y “Aceptar”.



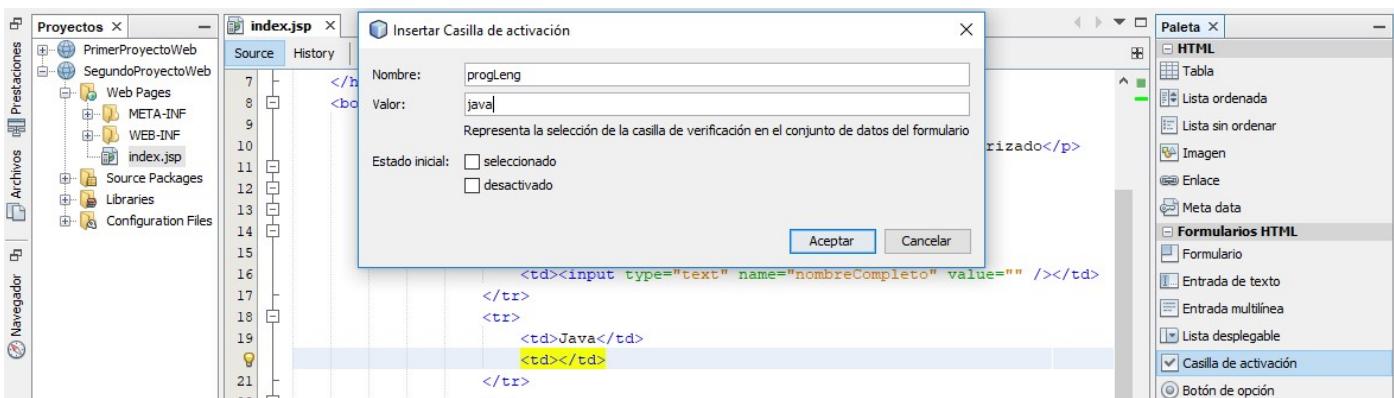
Dentro del “<form>” arrastramos una “Tabla”, de “6 Filas”, “2 Columnas”, “Borde 0”, y damos en “Aceptar”.



La tabla, crea por defecto la cabecera con “`<thead>`”, que no necesitamos y borramos. Rellenamos la primera fila con el texto “Nombre Completo” y un elemento de “Entrada de texto”, se llamará “nombreCompleto” y “Aceptar”.



En la siguiente fila, ponemos el primer lenguaje a elegir, “Java”, y después arrastramos una “Casilla de activación” para que lo podamos seleccionar o no. Introducimos de nombre “progLeng”, de valor “java” y “Aceptar”.

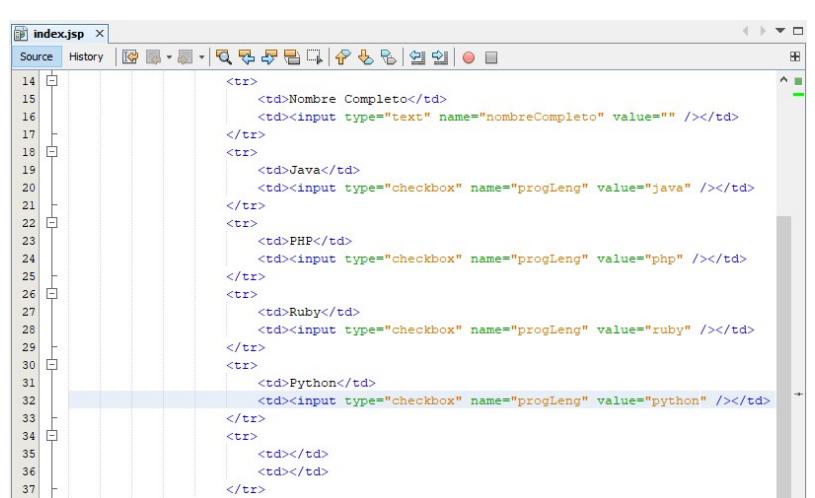


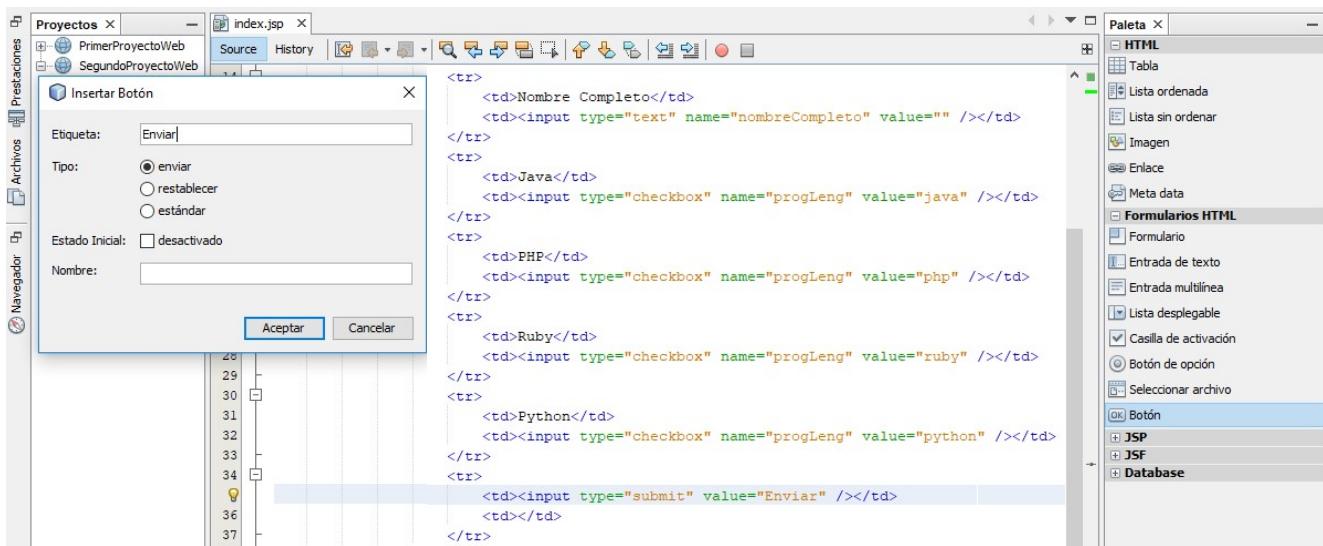
Ahora completamos el resto de lenguajes, “PHP”, “Ruby”, “Python”.

El elemento de casilla de activación la podemos copiar y cambiar el valor de cada opción que falta, poniendo “php”, “ruby” y “python”, dejando el mismo nombre “progLeng” en todas.

Al tener todas las casillas el mismo nombre, la información será enviada como un array de string, para que el fichero salida.jsp pueda recibir los datos.

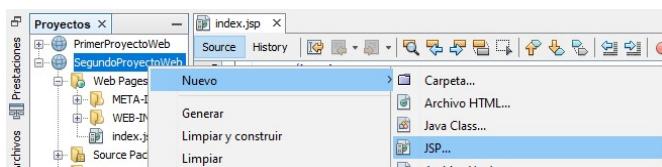
Falta arrastrar desde la paleta un “Botón”, poner la etiqueta “Enviar” y pulsar “Aceptar”, para completar este fichero.



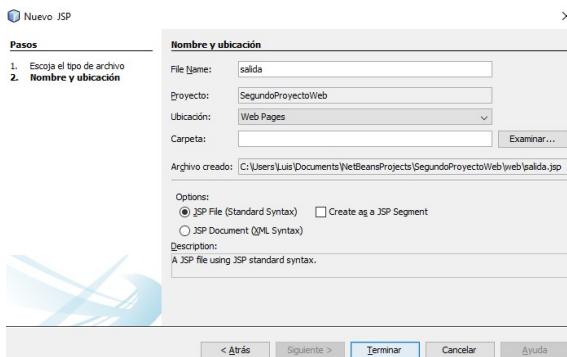


Podemos probar el proyecto y completar el formulario, pero si le damos a “Enviar” aparecerá un error ya que no tenemos todavía el fichero de “salida.jsp”.

Creamos este fichero seleccionando el proyecto, “SegundoProyectoWeb”, botón derecho, y en el menú contextual, elegimos “Nuevo” y “JSP”.



Cómo nombre del fichero ponemos “salida”, y al pulsar “Terminar” tendremos el contenido por defecto.



Modificamos el contenido por el que se muestra a la derecha. Se pueden ver con un fondo distinto y azulado, tanto las expresiones como los scriptlets

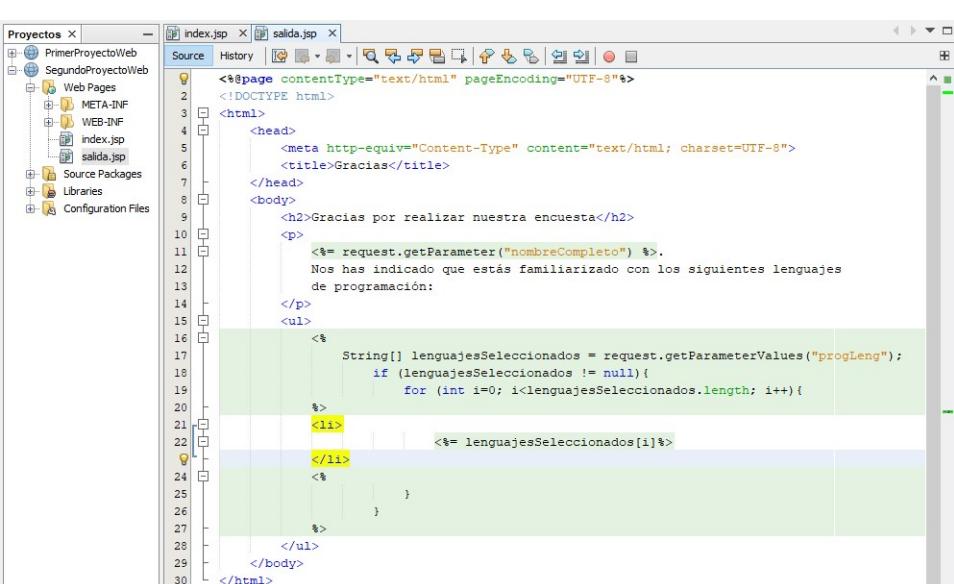
Recuperamos el valor de “nombreCompleto”

introducido. Guardamos en el string

“lenguajesSeleccionados” el objeto array que recuperamos, con las casillas de verificación.

Si la variable no está vacía, recorremos el array y mostramos cada elemento en forma de lista.

Terminamos cerrando llaves.



Ahora que ya hemos terminado el proyecto, podemos enviar el formulario completado para que el fichero “**salida.jsp**” muestre los datos que hemos introducido.

Gracias por realizar nuestra encuesta

Luis. Nos has indicado que estás familiarizado con los siguientes lenguajes de programación:

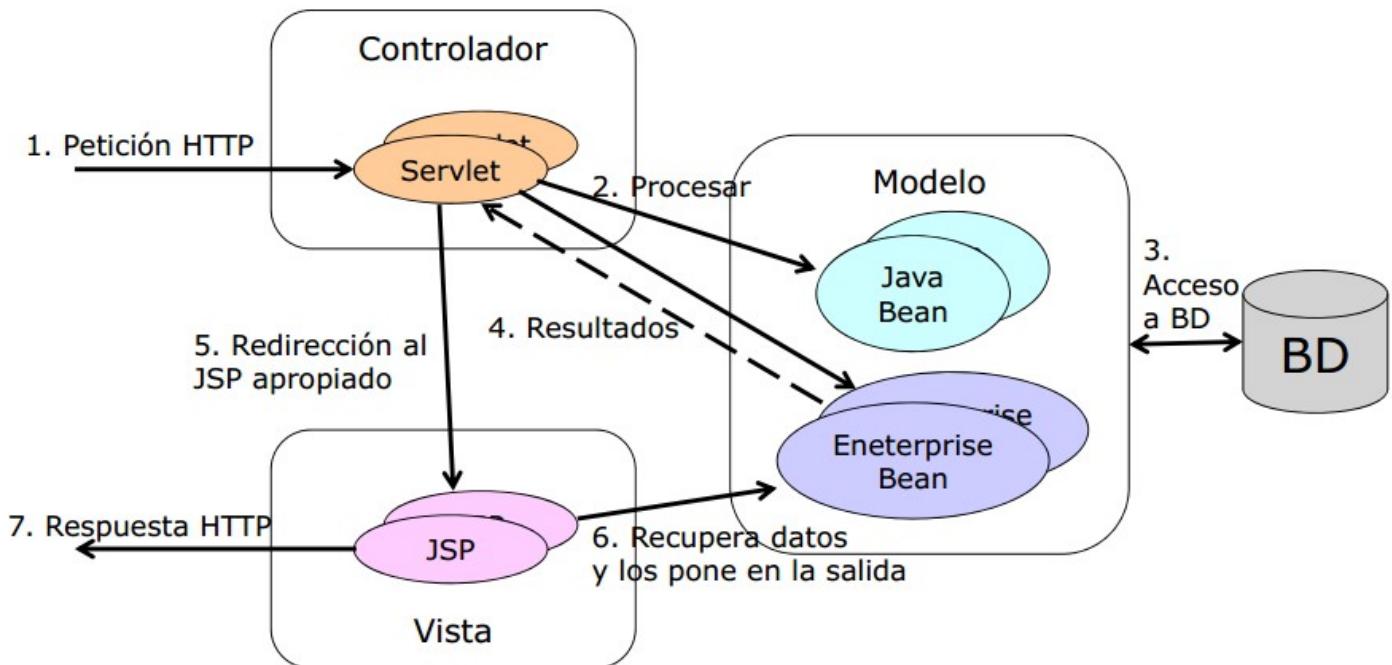
- java
- php

6 Java EE – Arquitectura MVC

6.1 Servlets, JSP y Enterprise beans

- Los **Servlets** facilitan el tratamiento de las peticiones que llegan al servidor
 - Tratamiento de datos de formularios
 - Generación de contenidos de formato variable
 - Permiten redireccionar las peticiones
- Las **JSP** facilitan el desarrollo y mantenimiento del contenido HTML
 - Interesante para páginas de formato establecido (poca variabilidad)
- Los **Java Beans** y **Enterprise Beans** facilitan la implementación de la lógica de negocio
 - Independiente del protocolo de interacción con los clientes
 - Independiente de la presentación de los resultados

6.2 Arquitectura MVC en Java EE



- Los beans representan los datos
 - Los resultados pueden ser Java Beans que encapsulan los resultados
 - Beans y Enterprise Beans pueden usarse para la lógica de negocio
- Los servlets gestionan las peticiones de los clientes
 - Reciben las peticiones HTTP
 - Procesan los parámetros (p.ej. de formularios) comprobando que son correctos
 - Invocan operaciones en beans para ejecutar la lógica de negocio
 - Guardan referencias a los beans de resultados en el ServletsContext, en la sesión o en la petición
- Reenvían la petición a una página JSP
 - El servlet determina la página JSP apropiada y usa el método de reenvío del RequestDispatcher para transferirle el control
- La página JSP accede a los beans de resultado
 - La página JSP no crea ni modifica beans, solo los consulta para ver los resultados

6.1 Servlets – Reenvío de peticiones

- El servlet determina la página JSP apropiada y usa el método de reenvío del RequestDispatcher para transferirle el control

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
    // Comprueba los parámetros de la petición (p.ej. de formularios)
    // Invoca operaciones en beans para ejecutar la lógica de negocio
    // Guarda referencias a los beans de resultados en
    // el ServletsContext, en la sesión o en la petición

    // Reenvía la petición a un JSP (Vista) para que haga la salida
    String operacion = request.getParameter("operacion");
    if (operacion == null) operacion = "desconocida";
    String jsp;
    if (operacion.equals("consulta"))    jsp = "/WEB-INF/Consulta.jsp";
    else if (operation.equals("cancel")) jsp = "/WEB-INF/Cancel.jsp";
    else      jsp = "/WEB-INF/Desconocida.jsp";

    RequestDispatcher dispatcher = request.getRequestDispatcher(jsp);
dispatcher.forward(request, response);
}

```

6.2 Paso de beans

- Se pueden pasar los beans por tres lugares
 - En el objeto request (HttpServletRequest)
 - Visibles para el servlet y la página o servlet a la que se lo reenvía
 - En el servlet:

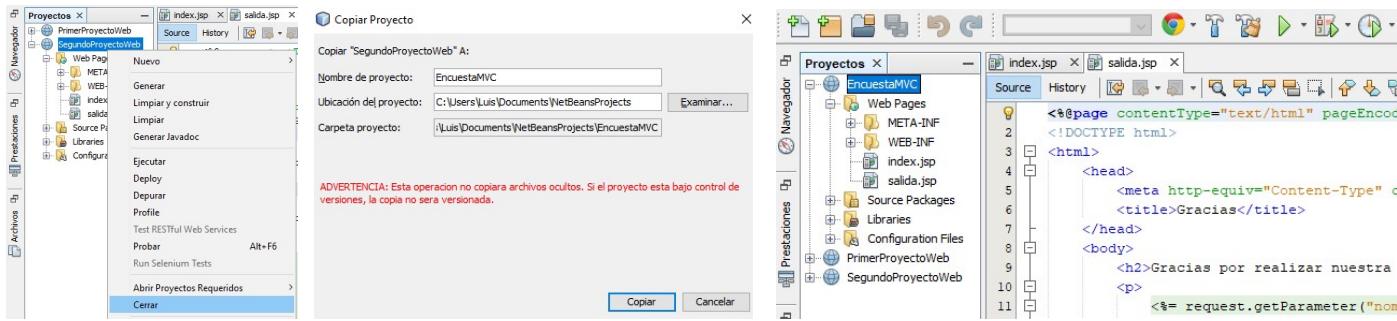
```
UnBean valor = LookupService.findResult(...);  
request.setAttribute("clave", valor);
```
 - En la página JSP:
 `${clave.propiedad}`
 - En la sesión (HttpSession)
 - Visibles para el servlet y la página o servlet a la que se lo reenvía, y para otras páginas posteriores referentes al mismo usuario
 - En el servlet, similar a antes pero:
 `session.setAttribute("clave", valor);`
 - En el Servlet Context
 - Son visibles para todos los servlets y páginas de la aplicación
 - En el servlet, similar a antes pero:
 `getServletContext().setAttribute("clave", valor);`
 - En otros servlets se tendrá que usar de forma sincronizada (synchronized)

6.3 La vista: páginas JSP

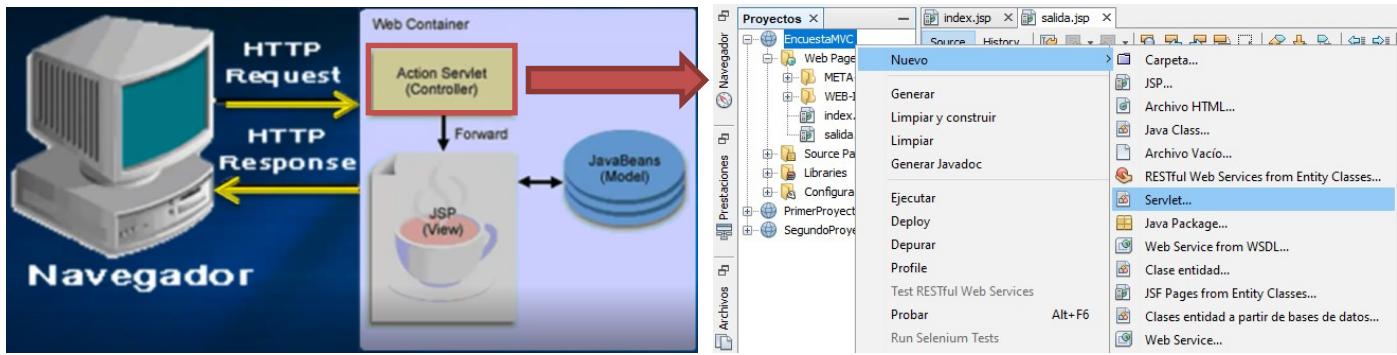
- La página JSP no crea beans
 - Para garantizar que no se crean beans, usar
`<jsp:useBean ... type="paquete.Clase" />`
 - en vez de
`<jsp:useBean ... class="paquete.Clase" />`
- La página JSP no modifica beans, solo los consulta para ver los resultados
 - Usar únicamente `jsp:getProperty`
 - No usar `jsp:setProperty`

7 Encuesta MVC (EncuestaMVC)

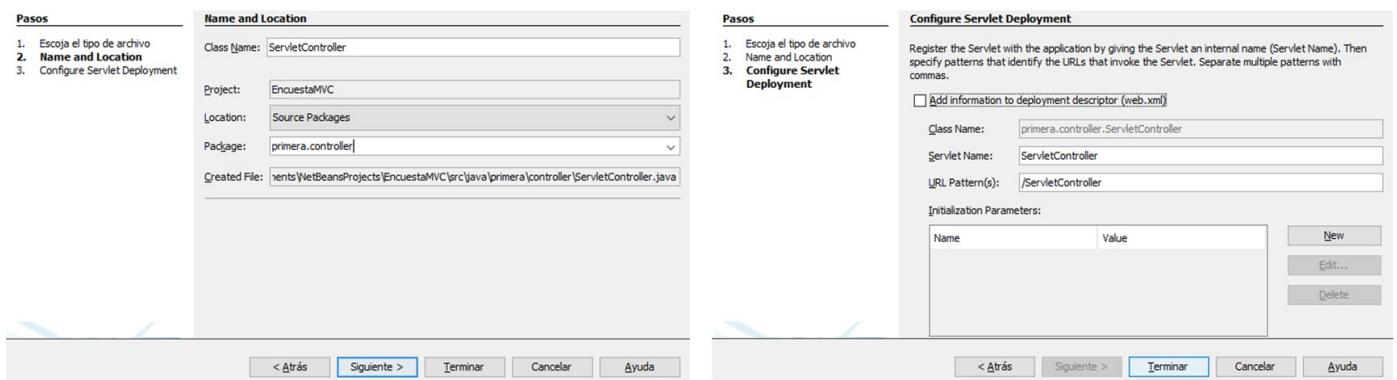
Aprovechamos el proyecto anterior “**SegundoProyectoWeb**”, para realizar una copia y actualizarlo usando MVC. Seleccionamos el nombre, y desde el menú contextual, elegimos “**Copiar**”. Cambiamos el nombre y ponemos “**EncuestaMVC**” y pulsamos el botón “**Copiar**”. Ahora podemos cerrar todos los ficheros anteriores, y comprimir su rama de contenidos. Seleccionamos el nombre del proyecto que acabamos de copiar, volvemos a poner el Chrome de navegador y abrimos los ficheros “index.jsp” y “salida.jsp”.



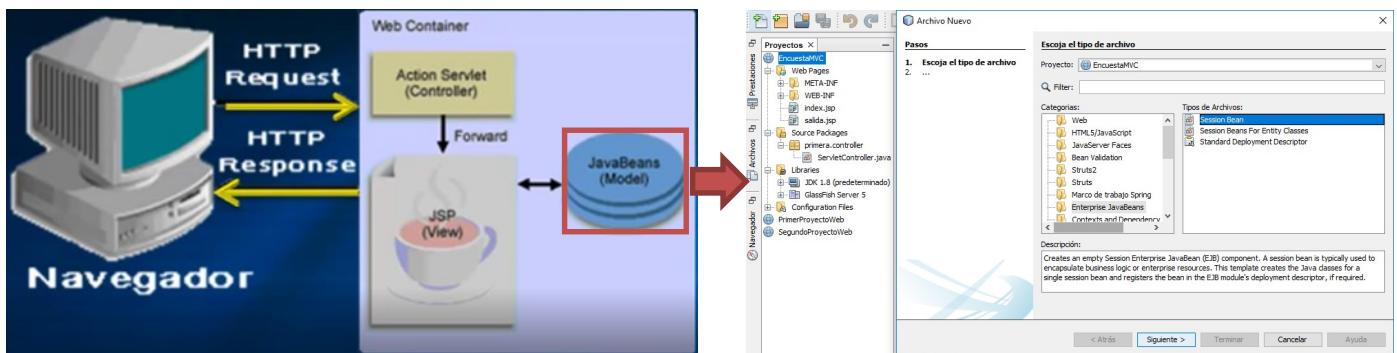
Adaptamos el proyecto para MVC comenzando por el **Controlador**, creamos un nuevo fichero señalando el nombre del proyecto, “**EncuestaMVC**”, desde el menú contextual, elegimos “**Nuevo**” y luego “**Servlet**”.



Ponemos “**ServletController**” y “**primera.controller**”, como nombres de clase y paquete respectivamente y pulsamos en “**Siguiente**”. En la siguiente pantalla, dejamos las opciones por defecto y pulsamos “**Terminar**”.



Seguimos adaptando el proyecto para MVC, para el **Modelo**, creamos un nuevo fichero desde el botón de “**Archivo Nuevo**”, categoría “**Enterprise JavaBeans**”, la primera opción de al lado “**Session Bean**”, y pulsamos “**Siguiente**”.



Ponemos “**DatosEncuesta**” y “**primera.modelo**”, como nombres del bean y paquete y “**Terminar**”. Borramos lo que no necesitamos, creamos las variables privadas e incluimos los getters y setters.

The screenshot shows the 'Nuevo Session Bean' wizard. In the 'Name and Location' step, the EJB Name is set to 'DatosEncuesta', the Project is 'EncuestaMVC', the Location is 'Source Packages', and the Package is 'primera.modelo'. The Session Type is selected as 'Stateless'. The 'Create Interface' checkbox is unchecked. The 'Terminar' button is highlighted.

The screenshot shows the generated Java code for the DatosEncuesta.java class. It defines a private String variable 'nombreCompleto' and a private String[] variable 'progLeng'. It includes getters and setters for both, and methods to return the array and set it.

Modificamos el “**ServletController**” añadiendo la importación del modelo que acabamos de crear, y sustituyendo el contenido del método “**processRequest**”. Se muestra imágenes del contenido anterior y cómo se tiene que quedar.

The screenshot shows the ServletController.java file. The processRequest method is highlighted with a red box. Inside the method, the code for writing HTML content is removed, and instead, it calls a new method 'datosEncuesta' which is defined in the code block below.

The screenshot shows the same file with the 'datosEncuesta' method highlighted. This method creates a new DatosEncuesta object, sets its properties from request parameters, and then forwards the request to 'salida.jsp'.

Una vez que tenemos el **Controlador** y el **Modelo**, es el turno de la **Vista**, referenciada por los **JSP**.

Modificamos el “**index.jsp**” cambiando el action del formulario “**salida.jsp**” por “**ServletController**” y añadiendo el método post. Se muestra imágenes del contenido anterior y cómo se tiene que quedar.

The screenshot shows the index.jsp file. The form tag has its action attribute changed from 'salida.jsp' to 'ServletController'. The method attribute is also set to 'post'.

Modificamos “**salida.jsp**” arrastrando de la paleta un “**Use Bean**” y un “**Get Bean Property**”, ademas de cambiar la asignación del string de los lenguajes. Se muestra imágenes del contenido anterior y nuevo.

The screenshot shows the salida.jsp file. It contains JSTL code for displaying selected languages. A 'Get Bean Property' dialog is open, showing 'Bean Name: DatosEncuesta' and 'Property Name: nombreCompleto'. The 'Aceptar' button is highlighted.

The screenshot shows the same file with the JSTL code updated. Instead of using the 'Get Bean Property' dialog, a 'Use Bean' component is used. The 'Bean' dropdown is set to 'DatosEncuesta', 'Class' to 'primera.modelo.DatosEncuesta', and 'Scope' to 'request'. The 'Aceptar' button is highlighted.

Podemos ejecutar el proyecto, comprobando que funciona exactamente igual que antes, utilizando MVC.

8 JSP Standard Tag Library (JSTL)

La tecnología JavaServer Pages Standard Tag Library (JSTL) es un componente de Java EE que extiende las ya conocidas JavaServer Pages (JSP) con utilidades ampliamente utilizadas en el desarrollo de páginas web dinámicas.

Son un conjunto de etiquetas organizadas en 5 categorías distintas, las cuales tienen como propósito fundamental evitar se introduzca código Java (scriptlets) en las páginas JSP. Los scriptlets son una manera desorganizada de programar páginas JSP y además, la simplicidad de las librerías que provee JSTL es de gran ayuda en la lógica de presentación de la página JSP.

CATEGORÍA	DESCRIPCIÓN
Etiquetas básicas	Son las etiquetas JSTL más comúnmente utilizados.
Etiquetas de SQL	Son para interactuar con una base de datos relacional.
Etiquetas de formato	Son para formatear el texto de salida, fecha, hora, número.
Las etiquetas XML	Son para crear y manipular documentos XML.
Función JSTL	Son funciones estándar, la mayoría de las cuales son funciones genéricas de manejo de cadenas.

Cada categoría, requiere de una directiva TagLib (Tag Libraries) que habrá que incluir al principio de cada archivo.

CATEGORÍA	TAGLIB NECESARIO
Etiquetas básicas	<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
Etiquetas de SQL	<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
Etiquetas de formato	<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
Las etiquetas XML	<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
Función JSTL	<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

Nosotros tan solo veremos las dos primeras categorías, “**Etiquetas básicas**” y “**Etiquetas de SQL**”, mostrando sus etiquetas y prácticando con algunas de ellas.

ETIQUETAS BÁSICAS	DESCRIPCIÓN
<c:out>	Permite mostrar valores en nuestra página.
<c:set>	Permite crear una variable y establecer un valor para esta.
<c:remove>	Borra el contenido de una variable que hayamos definido.
<c:catch>	Permite capturar excepciones.
<c:if>	Lo mismo que la instrucción if de Java SE, evalúa una condición.
<c:choose>	Permite establecer varias condiciones en un mismo bloque.
<c:when>	Para definir alguna condición de la etiqueta <c:choose>.
<c:otherwise>	Para definir el caso por defecto de la etiqueta <c:choose>.
<c:import>	Permite incluir contenido de otra página que se le indique.
<c:forEach>	Permite hacer iteraciones en una página jsp.
<c:forTokens>	Permite iterar sobre un String y separarlo por tokens.
<c:url>	Permite establecer una nueva url en la página que se le indique
<c:param>	Permite enviar una variable y su valor, por ejemplo, con la etiqueta anterior <c:url>
<c:redirect>	Permite redirigir a una página que se le indique.

ETIQUETAS DE SQL	DESCRIPCIÓN
<sql:setDataSource>	Para configurar la información del origen de datos o fuente de datos.
<sql:query>	Para ejecutar instrucción SQL SELECT.
<sql:update>	Para ejecutar una instrucción SQL que no devuelve valor (INSERT, UPDATE, DELETE).
<sql:param>	Para proporcionar un marcador de posición de valor.
<sql:dateParam>	Para proporcionar marcadores de posición de fecha y hora.
<sql:transaction>	Para empaquetar transacción.

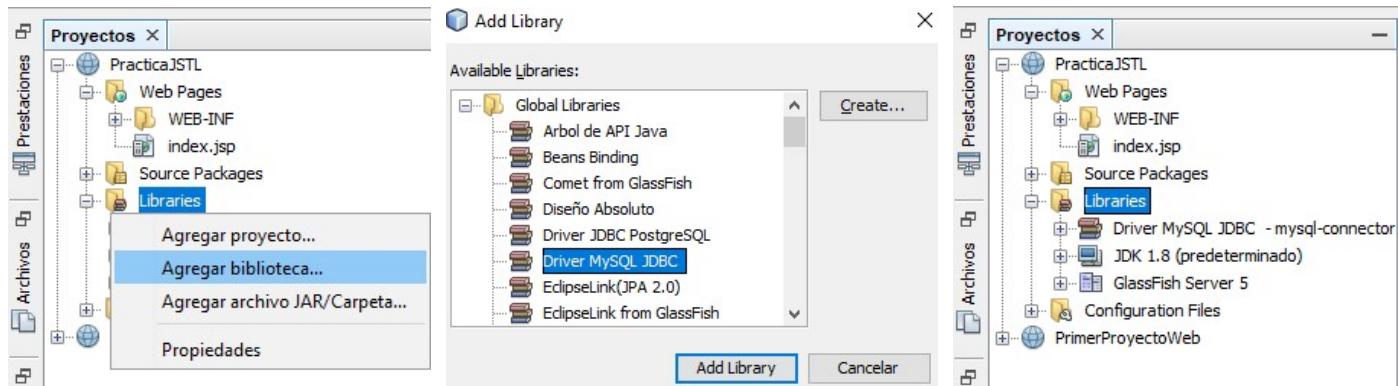
9 Conexión MySQL con JSTL (PracticaJSTL)

Vamos a trabajar con la base de datos “practicas”, y con una tabla “usuarios”.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	Ninguna		AUTO_INCREMENT	Cambiar Eliminar Más
2	nombre	varchar(100)	utf8_spanish_ci		No	Ninguna			Cambiar Eliminar Más
3	correo	varchar(100)	utf8_spanish_ci		No	Ninguna			Cambiar Eliminar Más

Como se ha realizado antes, desde Netbeans, copiamos “PrimerProyectoWeb” con el nombre “PracticaJSTL”.

Antes de poder trabajar con base de datos, tendremos que añadir la librería, señalando “Libraries”, y con el menú contextual señalamos “Agregar biblioteca”. Elegimos “Driver MySQL JDBC” en la lista y pulsamos “Add Library”.



Ahora modificamos el “index.jsp” añadiendo al principio los taglib para “Etiquetas básicas” y “Etiquetas de SQL”. Modificamos el título poniendo “Conexión MySQL con JSTL”.

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Conexión MySQL con JSTL</title>
    </head>
```

Creamos una variable en JSTL con “<c:set>” llamada “texto”, cuyo valor sea “Conexión MySQL con JSTL”. Después mostramos el contenido de la variable “texto” entre la cabecera de “<h2>” con “<c:out>”.

```
10 <body>
11     <c:set var="texto" value="Conexión MySQL con JSTL" />
12     <h2><c:out value="${texto}" /></h2>
```

Configuramos el acceso a la base de datos con “<sql:setDataSource>”.

```
13     <sql:setDataSource driver="com.mysql.jdbc.Driver"
14         url="jdbc:mysql://localhost/practicas"
15         user="root" password=""
16         var="con"/>
```

Le indicamos la consulta con “<sql:query>”, donde le pondremos la SELECT correspondiente.

```

17 |     <sql:query var="filas" dataSource="${con}">
18 |         SELECT * FROM usuarios;
19 |     </sql:query>

```

Para mostrar la información de la consulta utilizaremos una tabla de “1 fila” por “3 columnas” y “border=1”. Rellenamos la cabecera de la tabla con los nombres de los campos a mostrar “ID”, “NOMBRE” y “CORREO”.

```

20 |         <table border="1">
21 |             <thead>
22 |                 <tr>
23 |                     <th>ID</th>
24 |                     <th>NOMBRE</th>
25 |                     <th>CORREO</th>
26 |                 </tr>
27 |             </thead>

```

Utilizamos “” para recuperar los registros de la consulta, utilizando la variable “fila” para guardar cada registro recuperado del conjunto indicado por “\${filas.rows}”.

```

28 |         <tbody>
29 |             <c:forEach var="fila" items="${filas.rows}">

```

Por último, utilizamos “” para mostrar cada campo en su celda correspondiente y cerramos la etiqueta “” y el resto de html.

```

30 |         <tr>
31 |             <td><c:out value="${fila.id}" /></td>
32 |             <td><c:out value="${fila.nombre}" /></td>
33 |             <td><c:out value="${fila.correo}" /></td>
34 |         </tr>
35 |     </c:forEach>
36 |     </tbody>
37 |     </table>
38 | </body>
39 | </html>

```

Antes de probar la ejecución del proyecto, deberemos tener activado el modulo “MySQL” del “XAMPP”.

El resultado de ejecutar el proyecto es el siguiente, teniendo en cuenta que en clase se utiliza el puerto “9999”.

The screenshot shows the XAMPP Control Panel v3.2.2 interface. The MySQL service is running, indicated by a green status bar. The browser window shows a JSTL page titled "Conexión MySQL con JSTL" displaying user data from the database:

ID	NOMBRE	CORREO
1	Luis	lestan@iesdoctorbalmis.com
2	Pablo	pablo@iesdoctorbalmis.com